A Formal Analysis of Error Detecting Codes Using ACL2

Shilpi Goel

shigoel@cs.utexas.edu

The University of Texas at Austin

Outline

Outline

Outline

Error Detecting Codes

- Error Detecting Codes (EDCs) detect errors that may be introduced when data is received at the destination from the source.

Terminology

- Sender - where the data is augmented with a computed tag (converted into a codeword)
- Receiver - where it is checked whether the received message is a legal codeword or not

Outline

Why Do We Need To Analyze EDCs using ACL2?

- Need to trust their correctness and know their limitations
- Come up with a general framework that can be used for proofs of the properties of all EDCs

What Does The Analysis Of EDCs Involve?

- Receiver's Point of View: the receiver's concern is to correctly detect whether the received codeword is legal or not.

- Analyzer's Point of View: the analyzer's concern is to determine what kinds of errors the EDC scheme can detect.

What Does The Analysis Of EDCs Involve?

- Soundness (Receiver's Point of View)
- Completeness (Receiver's Point of View)
- Strength (Analyzer's Point of View)

Soundness

An informal description of soundness of an EDC:

- Given an uncorrupted transmission of data, the error control scheme ought to be able to report that the received data is error-free.

- No Error Detected $\Rightarrow$ Received Codeword is Legal

Completeness

An informal description of completeness of an EDC:

- Given a corrupted transmission of data, the error control scheme ought to be able to report that the data is corrupted.
- Error Detected $\Rightarrow$ Received Codeword is Not Legal

More About Soundness

- Merely knowing that the received codeword is legal is not enough to guarantee that the transmission was error-free.
- We need to analyze the strengths and limitations of the EDC to state under *what* conditions can we know absolutely that the transmission was error-free when it is reported as error-free.

Strength

Informally, determining the strength of an EDC involves the specification of:

- The types of errors that the EDC can *always* detect
- Includes the analysis of the general robustness of the EDC like detecting burst errors or more specific analysis like detection of isolated two bit errors, etc

Outline

Formalization

In this section, we will formalize the concepts of soundness, completeness and strength of an error detecting scheme.

Notation Used

Term	**Description**
Det	predicate that detects whether the received message is a codeword
E	function that encodes a message
D	function that decodes a message
Env	the predicate which specifies under what environment the EDC works

Outline

Soundness

Let n be the number of bits in the data the sender encodes and m be the message received by the receiver.

If Env(n,m) holds and Det(m) is false, then there exists an m' such that D(m) = m' and E(m') = m.

Outline

Completeness

Let n be the number of bits in the data the sender encodes and m be the message received by the receiver.

If Env(n,m) holds and if Det(m) is true, then there exists no m' such that D(m) = m' and E(m') = m.

Outline

Strength

Let s be the encoded message sent by the sender and r be the message received by the receiver such that r is not equal to s. Errors(s,r) specifies the transformations s undergoes to become r such that r is not another legal codeword.

If Errors(s,r) holds, then Det(r) will be true.

Outline

Analysis of Some EDCs

- To arrive at a general framework for EDC analysis, one must examine some specific EDCs.
  - Even Parity Check
  - Weighted Checksum
  - Cyclic Redundancy Check (CRC)
- In particular, we will look at the Soundness and Completenss of Cyclic Redundancy Checks.

Outline

CRCs

- CRCs are cyclic linear codes based on division in the ring of polynomials over GF(2).
- CRC division, like ordinary long division, can be done by shifts and subtraction (over GF(2)).

GF(2)

- Addition/Subtraction: XOR Operation

+	1	0
1	0	1
0	1	0

- Multiplication: AND Operation

*	1	0
1	1	0
0	0	0

For CRCs to work, the sender and receiver agree on:

- the generator polynomial gp, which is the divisor in the GF(2) division process
- the augment a - the length of a is one less than the length of gp. a is augmented to the message at the sender's end
- the number of bits n of the message to be encoded at a time by the sender - and hence, also the number of bits of the received message to be decoded at a time by the receiver

For CRCs to work, the sender and receiver agree on:

- the generator polynomial $\mathtt{gp}$, which is the divisor in the GF(2) division process
- the augment $\mathtt{a}$ - the length of $\mathtt{a}$ is one less than the length of $\mathtt{gp}$. $\mathtt{a}$ is augmented to the message at the sender's end
- the number of bits $\mathtt{n}$ of the message to be encoded at a time by the sender - and hence, also the number of bits of the received message to be decoded at a time by the receiver

For CRCs to work, the sender and receiver agree on:

- the generator polynomial $\mathtt{gp}$, which is the divisor in the GF(2) division process
- the augment $\mathtt{a}$ - the length of $\mathtt{a}$ is one less than the length of $\mathtt{gp}$. $\mathtt{a}$ is augmented to the message at the sender's end
- the number of bits $\mathtt{n}$ of the message to be encoded at a time by the sender - and hence, also the number of bits of the received message to be decoded at a time by the receiver

An Example

Let `gp` be 101 and `m'` be 1001. Hence, n is 4.

`a` should be of length 2.

Let `a` be 11.

An Example

Let **gp** be 101 and **m′** be 1001. Hence, n is 4.

a should be of length 2.

Let **a** be 11.

An Example

Let gp be 101 and m′ be 1001. Hence, n is 4.

a should be of length 2.

Let a be 11.

Sender's End

gp = 1 0 1
m′ = 1 0 0 1
a = 1 1

```
        _______________
1 0 1 ) 1 0 0 1 1 1 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 1 1
          0 0 0
        -----------
            1 1 1 1
            1 0 1
        -----------
              1 0 1
              1 0 1
        -----------
                0 0
```

00 is the computed tag (or CRC).

Sender's End

gp = 1 0 1
m′ = 1 0 0 1
a = 1 1

```
      ___________
1 0 1 ) 1 0 0 1 1 1 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 1 1
          0 0 0
        -----------
            1 1 1 1
            1 0 1
        -----------
              1 0 1
              1 0 1
        -----------
                0 0
```

00 is the computed tag (or CRC).

Sender's End

gp = 1 0 1
m′ = 1 0 0 1
a = 1 1

```
        ___________________
1 0 1 ) 1 0 0 1 1 1 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 1 1
          0 0 0
        -----------
            1 1 1 1
            1 0 1
        -----------
              1 0 1
              1 0 1
        -----------
                0 0
```

00 is the computed tag (or CRC).

Sender's End

gp = 1 0 1
m′ = 1 0 0 1
a = 1 1

```
        _______________
1 0 1 ) 1 0 0 1 1 1 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 1 1
          0 0 0
        -----------
            1 1 1 1
            1 0 1
        -----------
              1 0 1
              1 0 1
        -----------
                0 0
```

00 is the computed tag (or CRC).

Sender's End

$gp = 1\ 0\ 1$
$m' = 1\ 0\ 0\ 1$
$a = 1\ 1$

```
        _______________
1 0 1 ) 1 0 0 1 1 1 ( 1 0 1 1
        1 0 1
        ------------
          0 1 1 1 1
          0 0 0
        ------------
            1 1 1 1
            1 0 1
        ------------
              1 0 1
              1 0 1
        ------------
                0 0
```

00 is the computed tag (or CRC).

Sender's End

gp = 1 0 1
m′ = 1 0 0 1
a = 1 1

```
        _______________
1 0 1 ) 1 0 0 1 1 1 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 1 1
          0 0 0
        -----------
            1 1 1 1
            1 0 1
        -----------
              1 0 1
              1 0 1
        -----------
                0 0
```

00 is the computed tag (or CRC).

Receiver's End

Uncorrupted Transmission

```
        _______________
1 0 1 ) 1 0 0 1 0 0 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 0 0
          0 0 0
        -----------
            1 1 0 0
            1 0 1
        -----------
              1 1 0
              1 0 1
        -----------
                1 1
```

11 is the computed tag (or CRC).

Receiver's End

Uncorrupted Transmission

```
       ___________________
1 0 1 ) 1 0 0 1 0 0 ( 1 0 1 1
        1 0 1
        ------------
          0 1 1 0 0
          0 0 0
        ------------
            1 1 0 0
            1 0 1
        ------------
              1 1 0
              1 0 1
        ------------
                1 1
```

11 is the computed tag (or CRC).

Receiver's End

Uncorrupted Transmission

```
                ____________________
1 0 1 ) 1 0 0 1 0 0 ( 1 0 1 1
        1 0 1
        -----------
          0 1 1 0 0
          0 0 0
        -----------
            1 1 0 0
            1 0 1
        -----------
              1 1 0
              1 0 1
        -----------
                1 1
```

11 is the computed tag (or CRC).

Receiver's End

Corrupted Transmission

Instead of `1 0 0 1 0 0`, the receiver receives `1 1 0 1 0 0`.

```
        _______________
1 0 1 ) 1 1 0 1 0 0 ( 1 1 1 0
        1 0 1
        ------------
          1 1 1 0 0
          1 0 1
        ------------
            1 0 0 0
            1 0 1
        ------------
              0 1 0
              0 0 0
        ------------
                1 0
```

`1 0` is not equal to `a`.

Receiver's End

Corrupted Transmission

Instead of `1 0 0 1 0 0`, the receiver receives `1 1 0 1 0 0`.

```
      ____________________
1 0 1 ) 1 1 0 1 0 0 ( 1 1 1 0
        1 0 1
        -----------
          1 1 1 0 0
          1 0 1
        -----------
            1 0 0 0
            1 0 1
        -----------
              0 1 0
              0 0 0
        -----------
                1 0
```

`1 0` is not equal to `a`.

Receiver's End

Corrupted Transmission

Instead of `1 0 0 1 0 0`, the receiver receives `1 1 0 1 0 0`.

```
        _______________
1 0 1 ) 1 1 0 1 0 0 ( 1 1 1 0
        1 0 1
        -----------
          1 1 1 0 0
          1 0 1
        -----------
            1 0 0 0
            1 0 1
        -----------
              0 1 0
              0 0 0
        -----------
                1 0
```

`1 0` is not equal to `a`.

ACL2 Definitions

The Environment Predicate:

```
(defun Env (n m a gp)
  (and (<= 1 (len gp))
       (natp n)
       (< 0 n)
       (equal (car gp) 'T)
       (equal (len m) (+ n (len a)))
       (equal (len a) (1- (len gp)))
       (boolean-listp gp)
       (boolean-listp m)
       (boolean-listp a)))
```

ACL2 Definitions

The Detecting Predicate:

```
(defun Det (m a gp)
  (not (equal (crc m gp) a)))
```

Encoding Function:

```
(defun E (m- a gp)
  (append m-
          (crc (append m- a) gp)))
```

Decoding Function:

```
(defun D (m gp)
  (firstn (- (len m) (1- (len gp)))
          m))
```

Soundness

```
(defun-sk exists-D-E (m a gp)
    (exists (m-)
            (and (equal (D m gp) m-)
                 (equal (E m- a gp) m))))
```

```
(defthm soundness-crc
    (implies (and (Env n m a gp)
                  (not (Det m a gp)))
             (exists-D-E m a gp)))
```

Completeness

```
(defthm completeness-crc
    (implies (and (Env n m a gp)
                  (Det m a gp))
             (not (exists-D-E m a gp))))
```

Strength Of CRCs - Work In Progress

```
(defun burst-error (s r gp)
  (< (len (strip-nils-at-ends (bv-xor s r))) (len gp)))
```

```
(defthm strength-of-crc
 (implies (and (Env n r a gp)
               (equal s (E m- a gp))
               (boolean-listp m-)
               (equal (len m-) n)
               (equal (car (last gp)) 'T)
               (burst-error s r gp))
          (Det r a gp)))
```

Outline

Summary

- As of now, we have done a formal analysis of soundness, completeness and strengths of some EDCs.
- We aim to arrive at a general framework to prove the correctness and strengths of all EDCs.
- Doing a similar analysis for Error Correcting Codes would be interesting too.